Software specification
for
Kontron EAPI

Version *1.21*

Author: Martin Kroul
Last change date: 26.4.2011

Contents

CONTENTS 2

1 INTRODUCTION 5

1.1 Purpose of this document 5

1.2 KEAPI overview 5

1.3 Architectural overview 5

1.4 Operating systems 6

2 FUNCTION DOCUMENTATION 6

2.1 Return values 6
2.1.1 Functional errors 6
2.1.2 Remote connection errors 6

2.2 Initialization 7
2.2.1 KEApiLibInitialize 7
2.2.2 KEApiLibUnInitialize 7

2.3 General information 7
2.3.1 KEApiGetBoardInfo 7
2.3.2 KEApiGetBootCounter 8
2.3.3 KEApiSystemUpTime 8
2.3.4 KEApiGetSystemStateAmt 9
2.3.5 KEApiSetSystemStateAmt 9
2.3.6 KEApiSetSystemState 9
2.3.7 KEApiGetIntruderStatus 10
2.3.8 KEApiResetIntruderStatus 10

2.4 CPU 10
2.4.1 KEApiGetCpuFreq 10
2.4.2 KEApiGetCpuInfo 11
2.4.3 KEApiGetCpuPerformance 11
2.4.4 KEApiSetCpuPerformance 12

2.5 Memory 12
2.5.1 KEApiGetMemoryInfo 12
2.5.2 KEApiGetMemoryModuleCount 13
2.5.3 KEApiGetMemoryModuleInfo 13

2.6 Hard disks and partitions 14
2.6.1 KEApiGetDiskDriveCount 14
2.6.2 KEApiGetDiskDriveList 14
2.6.3 KEApiGetDiskDrivePartitionCount 14
2.6.4 KEApiGetDiskDrivePartitionList 15

2.7 Battery 15
2.7.1 KEApiGetBatteryCount 15

2.7.2 KEApiGetBatteryInfo 15
2.7.3 KEApiGetBatteryState 16

2.8 Temperature sensors 17
2.8.1 KEApiGetTempSensorCount 17
2.8.2 KEApiGetTempSensorList 17
2.8.3 KEApiGetTempSensorValue 17
2.8.4 KEApiGetCpuTemp 18
2.8.5 KEApiGetChipsetTemp 18
2.8.6 KEApiGetSystemTemp 18

2.9 Voltage sensors 19
2.9.1 KEApiGetVoltageSensorCount 19
2.9.2 KEApiGetVoltageSensorList 19
2.9.3 KEApiGetVoltageSensorValue 19

2.10 Fan Control 20
2.10.1 KEApiGetFanCount 20
2.10.2 KEApiGetFanList 20
2.10.3 KEApiGetFanSpeed 21
2.10.4 KEApiGetFanMode 21
2.10.5 KEApiSetFanModeManual 21
2.10.6 KEApiSetFanModeThermalCruise 22
2.10.7 KEApiGetCpuFanSpeed 22
2.10.8 KEApiSetCpuFanSpeed 22
2.10.9 KEApiGetSystemFanSpeed 23
2.10.10 KEApiSetSystemFanSpeed 23

2.11 Display 23
2.11.1 KEApiGetDisplayCount 23
2.11.2 KEApiGetBacklightValue 24
2.11.3 KEApiSetBacklightValue 24

2.12 Network and PCI devices 24
2.12.1 KEApiGetNetworkDeviceCount 24
2.12.2 KEApiGetNetworkDeviceList 25
2.12.3 KEApiGetPciDeviceCount 25
2.12.4 KEApiGetPciDeviceList 25

2.13 Storage area 26
2.13.1 KEApiGetStorageCount 26
2.13.2 KEApiGetStorageSize 26
2.13.3 KEApiStorageRead 27
2.13.4 KEApiStorageWrite 27

2.14 I2C 27
2.14.1 KEApiGetI2cBusCount 27
2.14.2 KEApiI2cWrite 28
2.14.3 KEApiI2cRead 28

2.15 SMBus 29
2.15.1 KEApiGetSmbusCount 29
2.15.2 KEApiSmbusQuickCommand 29
2.15.3 KEApiSmbusSendByte 29
2.15.4 KEApiSmbusReceiveByte 30
2.15.5 KEApiSmbusWriteByte 30
2.15.6 KEApiSmbusReadByte 31

2.15.7 KEApiSmbusWriteWord 31
2.15.8 KEApiSmbusReadWord 32
2.15.9 KEApiSmbusWriteBlock 32
2.15.10 KEApiSmbusReadBlock 32

2.16 GPIO 33
2.16.1 KEApiGpioGetCount 33
2.16.2 KEApiGpioGetDirectionCaps 33
2.16.3 KEApiGpioGetDirection 34
2.16.4 KEApiGpioSetDirection 34
2.16.5 KEApiGpioGetLevel 34
2.16.6 KEApiGpioSetLevel 35

2.17 Watchdog 35
2.17.1 KEApiWatchdogEnable 35
2.17.2 KEApiWatchdogTrigger 35
2.17.3 KEApiWatchdogDisable 36

3 ABOUT KONTRON 37

Revision history

Author	Date	Change summary	Version
Martin Kroul	*20.6.2010*	*Document created*	*1.0*
Martin Kroul	*5.7.2010*	*Major changes in functionality according to conf. call 30.6.2010*	*1.1*
Martin Kroul	*28.1.2011*	*First official release of KEAPI for Windows and Linux* *- restructuralization of function names, parameters and types* *- no changes in functionality*	*1.2*
Martin Kroul	*26.4.2011*	*VxWorks added to list of supported operating systems*	*1.21*

Legal notice:
All data is for information purposes only and not guaranteed for legal purposes. Subject to change without notice. Information in this datasheet has been carefully checked and is
believed to be accurate; however, no responsibility is assumed for inaccuracies. All brand or product names are trademarks or registered trademarks of their respective owners.

1 Introduction

1.1 Purpose of this document

This document describes Kontron Embedded API – a software library that enables programmers to easily create their applications for monitoring and control hardware resources of Kontron boards, modules, systems and platforms.

1.2 KEAPI overview

KEAPI is written in C language and distributed as a dynamic-linked library so it can be used by arbitrary application developed in C, C++ or higher programming languages simply by linking to the project.

KEAPI library provides a set of functions for

- Obtaining basic information about the system
- CPU performance control
- Temperature and voltage sensors monitoring
- Monitoring and control of fans
- Display backlight control
- Access to EEPROM user storage area
- I2C and SMBus communication
- GPIO control
- Watchdog

All KEAPI functions can be called remotely from another computer over network (TCP/IP).

KEAPI is compatible with PICMG EAPI specification (http://www.picmg.org/pdf/COM_EAPI_R1_0.pdf). This means that it implements all functions defined by this specification. However, these functions offer just a small part of KEAPI functionalities and don't allow remote connection.

1.3 Architectural overview

Kontron EAPI is a layer between Kontron hardware and user application. KEAPI functions either use hardware drivers and operating system calls to perform requested operation on local board or connects to KEAPI server that is installed on remote board somewhere on the internet. The KEAPI server application listens on a predefined port (default is 7890) and waits for incoming request. When the request comes, it uses KEAPI library to perform the operation and sends result back to the client.

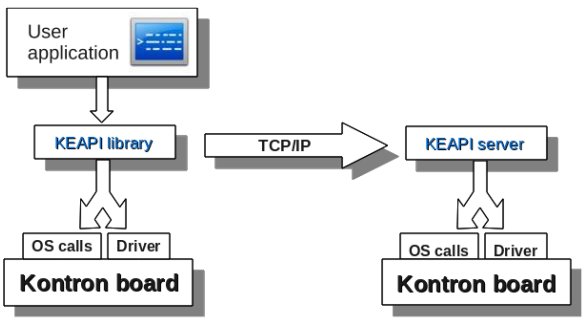

Fig.1. – Kontron EAPI Architecture

1.4 Operating systems

This document describes KEAPI interface for Linux, Windows XP, Windows Vista - 32/64, Windows 7 - 32/64, VxWorks.

2 Function Documentation

Main part of Kontron EAPI is dynamic-linked library with API functions. Before using any of these functions, KEAPI has to be initialized and connection to the board has to be established by calling the **KEApiLibInitialize()** function. This function returns a board handle (unique four-byte number) that will other KEAPI functions take as their first parameter. The **KEApiLibInitialize()** function may be called several times for simultaneous connection to several boards. When KEAPI is no longer needed, the **KEApiLibUnInitialize()** function should be called.

Memory for structures and variables that will be used by KEAPI functions to store information must be pre-allocated by the user application. KEAPI by itself doesn't allocate memory for output parameters.

2.1 Return values

If everything goes well, all KEAPI functions return KEAPI_RET_SUCCESS. If some error occurs, the returned error message differs for various functions. Error messages can be basically divided into two groups: Functionality errors and Connection errors.

2.1.1 Functional errors

KEAPI_RET_ERROR	General Error
KEAPI_RET_PARAM_ERROR	Wrong parameter value
KEAPI_RET_PARAM_NULL	Parameter is NULL (memory not pre-allocated)
KEAPI_RET_WD_TIMEOUT_NOT_SUPPORTED	WD timeout not supported
KEAPI_RET_BUFFER_OVERFLOW	Buffer overflow (probably configuration error)
KEAPI_RET_SETTING_ERROR	Error while setting value or feature (enable, disable)
KEAPI_RET_RETRIEVAL_ERROR	Error while retrieving information
KEAPI_RET_WRITE_ERROR	Cannot write to file
KEAPI_RET_READ_ERROR	Cannot read file
KEAPI_RET_MALLOC_ERROR	Memory allocation failed
KEAPI_RET_LIBRARY_ERROR	Exported function could not be loaded from library
KEAPI_RET_WMI_ERROR	Problems while reading from WMI
KEAPI_RET_NOT_INITIALIZED	KEAPI library is not initialized
KEAPI_RET_PARTIAL_SUCCESS	Part of requested information couldn't be retrieved. Returned information isn't complete.
KEAPI_RET_FUNCTION_NOT_SUPPORTED	Function is not supported on current platform/HW
KEAPI_RET_FUNCTION_NOT_IMPLEMENTED	Function is not yet implemented

2.1.2 Remote connection errors

KEAPI_RET_INVALID_IP	IP address is not in a valid format
KEAPI_RET_RMT_CON_ERROR	Remote connection failed
KEAPI_RET_RPC_ERR_MAX_CONN_COUNT_EXCEEDED	Connection limit on the client side was reached
KEAPI_RET_RPC_ERR_SOCKET_ERROR	Error with socket manipulation
KEAPI_RET_RPC_ERR_CONNECTION_NOT_FOUND	Connection for given board handle does not exist
KEAPI_RET_RPC_ERR_NOT_IMPLEMENTED	Remote function call is not implemented
KEAPI_RET_RPC_ERR_SOCKET_TIMEOUT	Time limit for select() function expired
KEAPI_RET_RPC_ERR_PACKET_ERROR	Wrong format of KEAPI packet

2.2 Initialization

2.2.1 KEApiLibInitialize

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiLibInitialize (
      KEAPI_PCHAR pHostIp,
      KEAPI_UINT32 port,
      KEAPI_PUINT32 pBoardHandle,
      KEAPI_UINT32 logLevel,
      KEAPI_PCHAR pLogFile
);
```

Description:

Initialization of Kontron EAPI.

Parameters:

in/out	Parameter name	Description
in	*pHostIp*	IP address of a remote computer. If it is set to NULL or "localhost" then localhost is considered to be the target machine (no KEAPI server running on the machine is needed). If it is set to "127.0.0.1", then it connects to KEAPI server running on the local machine (can be used for testing KEAPI server application locally)
in	*port*	Port number where KEAPI server is listening, default is 7890.
out	*pBoardHandle*	Returned pointer to the handle of selected board. *BoardHandle* == 0 means localhost.
In	*logLevel*	Level of details • LV_ERROR = 0x01, • LV_WARN = 0x02, • LV_INFO = 0x04, • LV_DEBUG = 0x08, • LV_TRACE = 0x10, • LV_ALLTIME = 0xFF
in	*pLogFile*	Name of the log file. Can be NULL if logging not needed.

2.2.2 KEApiLibUnInitialize

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiLibUnInitialize (
      KEAPI_UINT32 boardHandle
);
```

Description:

Uninitialization of Kontron EAPI on a selected board.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board

2.3 General information

2.3.1 KEApiGetBoardInfo

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetBoardInfo (
      KEAPI_UINT32 boardHandle,
      PBOARD_INFO pBoardInfo
);
```

Description:

Provides information about Kontron motherboard.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pBoardInfo*	Returned board info structure **BOARD_INFO**

Structure used:

```
typedef struct Board_Info
{
     KEAPI_CHAR  boardManufacturer[MAX_STR];  // Board manufacturer
     KEAPI_CHAR  boardName[MAX_STR];          // Board name
     KEAPI_CHAR  boardVersion[MAX_STR];       // Board version
     KEAPI_CHAR  boardSerialNumber[MAX_STR]; // Board serial number
     KEAPI_CHAR  biosVersion[MAX_STR];        // Bios version
     KEAPI_CHAR  biosDate[MAX_STR];           // BIOS date in format
dd/mm/yyyy
} BOARD_INFO, *PBOARD_INFO;
```

2.3.2 KEApiGetBootCounter

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetBootCounter (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pBootCount
);
```

Description:

Provides information about number of boot cycles within the board's lifetime.

Parameters:

in/out	Parameter name	Description
In	*boardHandle*	Handle of selected board
Out	*pBootCount*	Number of boot cycles

2.3.3 KEApiSystemUpTime

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSystemUpTime (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pSystemUpTime
);
```

Description:

Provides time left since last boot in seconds.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pSystemUpTime*	Pointer to a variable that receives system running time in seconds

2.3.4 KEApiGetSystemStateAmt

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetSystemStateAmt (
      KEAPI_PCHAR  pHostIp,
      KEAPI_PCHAR  pUser,
      KEAPI_PCHAR  pPassword,
      KEAPI_PINT32 pSystemState
);
```

Description:

Uses Intel AMT to get information about state of remote board (turned on, turned off or hibernated). The remote board must support Intel AMT and enable it in BIOS. AMT functionality can be tested via web browser (fill in address **http://** "*remote board's IP*" **:16992** and AMT web page should appear).

Parameters:

in/out	Parameter name	Description
in	*pHostIp*	IP address of the remote board
in	*pUser*	User name for host AMT
in	*pPassword*	Password for host AMT
out	*pSystemState*	Pointer to variable that will receive remote board's state • SYSTEM_STATE_TURN_ON, 0 • SYSTEM_STATE_TURN_OFF, 1 • SYSTEM_STATE_HIBERNATE, 2

2.3.5 KEApiSetSystemStateAmt

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetSystemStateAmt (
      KEAPI_PCHAR  pHostIp,
      KEAPI_PCHAR  pUser,
      KEAPI_PCHAR  pPassword,
      KEAPI_INT32  systemState
);
```

Description:

Uses Intel AMT to turn on, turn off or restart remote board. If the remote board doesn't support Intel AMT, then the *KEApiSetSystemState()* function using OS call can be used instead.

Parameters:

in/out	Parameter name	Description
in	*pHostIp*	IP address of the remote board
in	*pUser*	User name for host AMT
in	*pPassword*	Password for host AMT
in	*systemState*	Desired system state • SYSTEM_STATE_TURN_ON, 0 • SYSTEM_STATE_TURN_OFF, 1 • SYSTEM_STATE_RESTART, 3

2.3.6 KEApiSetSystemState

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetSystemState (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32  systemState
);
```

Description:

Uses operating system's command to turn on, turn off or restart remote board. There is no guarantee that the command was performed correctly i.e. if the system hangs.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*systemState*	Desired system state • SYSTEM_STATE_TURN_OFF, 1 • SYSTEM_STATE_HIBERNATE, 2 • SYSTEM_STATE_RESTART, 3

2.3.7 KEApiGetIntruderStatus

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetIntruderStatus (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pIntruderStatus
);
```

Description:

Provides actual information whether computer case was opened or not.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pIntruderStatus*	Pointer to a variable that receives actual intruder status • INTRUDER_STATUS_CASE CLOSED, 0 • INTRUDER_STATUS_CASE OPENED , 1

2.3.8 KEApiResetIntruderStatus

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiResetIntruderStatus (
      KEAPI_UINT32 boardHandle
);
```

Description:

Resets the case intruder status.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board

2.4 CPU

2.4.1 KEApiGetCpuFreq

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetCpuFreq (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT8 freqType,
      KEAPI_PINT32 frequency
);
```

Description:

Provides information about CPU frequency

- minimal supported CPU frequency: CPU_FREQUENCY_MIN, 0
- maximal supported CPU frequency: CPU_FREQUENCY_MAX, 1
- current CPU frequency: CPU_FREQUENCY_CURRENT, 2.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*freqType*	Type of frequency (minimal (CPU_FREQUENCY_MIN, 0), maximal (CPU_FREQUENCY_MAX, 1), current (CPU_FREQUENCY_CURRENT, 2))
out	*frequency*	Frequency of the CPU, in kHz

2.4.2 KEApiGetCpuInfo

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetCpuInfo (
      KEAPI_UINT32 boardHandle,
      PCPU_INFO pCpuInfo
);
```

Description:

Provides information about processors.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pCpuInfo*	Returned CPU_INFO structure

Structure used:

```
typedef struct Cpu_Info
{
     KEAPI_CHAR  cpuName[MAX_STR];  // CPU name
     KEAPI_INT32 cpuCount;          // Number of CPUs
     KEAPI_INT32 cpuCoreCount;      // Number of cores of each CPU
     KEAPI_INT32 cpuThreadCount;    // Number of CPU threads
     KEAPI_FLOAT cpuMaxCoreSpeed;   // Designed core speed in MHz
     KEAPI_INT32 fsbSpeed;          // FSB speed in MHz
     KEAPI_INT32 l1Cache;           // L1 cache size in kB
     KEAPI_INT32 l2Cache;           // L2 cache size in kB
     KEAPI_INT32 l3Cache;           // L3 cache size in kB
} CPU_INFO, *PCPU_INFO;
```

2.4.3 KEApiGetCpuPerformance

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetCpuPerformance (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT8 pPerformanceType,
      KEAPI_PINT8 pPerformancePercentage
);
```

Description:

Linux: Provides information about the current CPU performance mode

- high performance: CPU_PERFORMANCE_HIGH, 0
- power save mode: CPU_PERFORMANCE_POWER_SAVE, 1
- on demand mode: CPU_PERFORMANCE_ON_DEMAND, 2
- conservative mode: CPU_PERFORMANCE_CONSERVATIVE, 3.

Windows: Provides information about the current CPU performance in percentage

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pPerformanceType*	Pointer to the type of current CPU performance. The type is given by a number (0 – 3 where: • 0 - high performance • 1 - power save mode • 2 - on demand mode • 3 - conservative mode) . Supported only on Linux
out	*pPerformancePercentage*	Pointer to current CPU performance in percentage. Supported only on Windows

2.4.4 KEApiSetCpuPerformance

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetCpuPerformance (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT8 performanceType,
      KEAPI_INT8 performancePercentage
);
```

Description:

Linux: Allows you to set the type of the CPU performance

- high performance: CPU_PERFORMANCE_HIGH, 0
- power save mode: CPU_PERFORMANCE_POWER_SAVE, 1
- on demand mode: CPU_PERFORMANCE_ON_DEMAND, 2
- conservative mode: CPU_PERFORMANCE_CONSERVATIVE, 3.

Windows: Allows you to set CPU performance in percentage

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*performanceType*	CPU performance type Supported only on Linux
in	*performancePercentage*	CPU performance in percentage Supported only on Windows

2.5 Memory

2.5.1 KEApiGetMemoryInfo

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetMemoryInfo (
      KEAPI_UINT32 boardHandle,
      PMEMORY_INFO pMemoryInfo
);
```

Description:

Provides information about physical memory.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of selected board
out	*pMemoryInfo*	Returned MEMORY_INFO structure

Structure used:

```
typedef struct Memory_Info
{
     KEAPI_INT32 memTotal;        // Total physical memory size in MB
     KEAPI_INT32 memFree;         // Free memory in MB
     KEAPI_INT32 memSpeed;        // Memory speed in MHz
     KEAPI_CHAR  memType[MAX_STR];  // Type of memory (DDR, DDR2, etc.
} MEMORY_INFO, *PMEMORY_INFO;
```

2.5.2 KEApiGetMemoryModuleCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetMemoryModuleCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pMemoryModuleCount
);
```

Description:

Provides number of installed memory modules.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of selected board
out	*pMemoryModuleCount*	Number of installed memory modules

2.5.3 KEApiGetMemoryModuleInfo

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetMemoryModuleInfo (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT8 moduleNr,
      PMEMORY_MODULE pMemoryModule
);
```

Description:

Provides information about selected memory module.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*moduleNr*	Requested memory's module number. Numbers start with zero.
out	*pMemoryModule*	Returned MEMORY_MODULE structure

Structure used:

```
typedef struct MemoryModule
{
     KEAPI_INT32  moduleSize;             // Module capacity in MB
     KEAPI_CHAR   manufacturer[MAX_STR];  // Memory module manufacturer
     KEAPI_CHAR   partNumber[MAX_STR];    // Memory module part number
     KEAPI_CHAR   serialNumber[MAX_STR];  // Memory module serial number
} MEMORY_MODULE, *PMEMORY_MODULE;
```

2.6 Hard disks and partitions

2.6.1 KEApiGetDiskDriveCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetDiskDriveCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pDiskDriveCount
);
```

Description:

Provides number of installed disk drives.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pDiskDriveCount*	Number of installed disk drives

2.6.2 KEApiGetDiskDriveList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetDiskDriveList (
      KEAPI_UINT32 boardHandle,
      PDISK_DRIVE pDiskDrives,
      KEAPI_INT32 diskDriveCount
);
```

Description:

Provides list of disk drives and their properties.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*diskDriveCount*	Number of disks
out	*pDiskDrives*	Returned array of DISK_DRIVE structures. The array must be allocated as *DiskDriveCount* * sizeof(DISK_DRIVE), where *DiskDriveCount* is obtained from calling **KEApiGetDiskDriveCount**.

Structure used:

```
typedef struct Disk_Drive
{
     KEAPI_CHAR   name[MAX_STR];                // HDD name
     KEAPI_CHAR   model[MAX_STR];               // Model
     KEAPI_CHAR   diskSerialNumber[MAX_STR];    // Serial number
     KEAPI_UINT32 size;                         // Size in MB
} DISK_DRIVE, *PDISK_DRIVE;
```

2.6.3 KEApiGetDiskDrivePartitionCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetDiskDrivePartitionCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pDiskPartitionCount
);
```

Description:

Provides number of partitions.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pDiskPartitionCount*	Number of partitions

2.6.4 KEApiGetDiskDrivePartitionList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetDiskDrivePartitionList (
      KEAPI_UINT32 boardHandle,
      PDISK_PARTITION pDiskPartitionData,
      KEAPI_UINT32 diskPartitionCount
);
```

Description:

Provides list of disk partitions.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*diskPartitionCount*	Number of partitions (size of elements in the *pDiskPartitionData* array)
out	*pDiskPartitionData*	Array of **DISK_PARTITION** structures

Structure used:

```
typedef struct Disk_Partition
{
     KEAPI_CHAR     name[MAX_STR];   // Partition name
     KEAPI_CHAR     fsType[MAX_STR]; // Filesystem type
     KEAPI_UINT32   partitionSize;   // Size in MB
     KEAPI_UINT32   freeSpace;       // Free space in MB
} DISK_PARTITION, *PDISK_PARTITION;
```

2.7 Battery

2.7.1 KEApiGetBatteryCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetBatteryCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pBatteryCount
);
```

Description:

Provides number of connected batteries.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pBatteryCount*	Number of connected batteries

2.7.2 KEApiGetBatteryInfo

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetBatteryInfo (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT8 batteryNr,
      PBATTERY_INFO pBatteryInfo
);
```

Description:

Provides information about selected battery.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*batteryNr*	Requested battery's number. Numbers start with zero.
out	*pBatteryInfo*	Returned BATTERY_INFO structure

Structure used:

```
typedef struct Battery_Info
{
     KEAPI_CHAR  deviceName[MAX_STR];   // Device name
     KEAPI_CHAR  type[MAX_STR];         // LION, NiCd, NiMH…
     KEAPI_CHAR  serialNumber[MAX_STR]; // Serial number
     KEAPI_INT32 designedVoltage;       // Designed voltage in mV
     KEAPI_INT32 designedCapacity;      // Designed capacity of fully
charged battery in mAh
     KEAPI_INT32 fullyChargedCapacity;  // Real capacity of fully charged
battery in mAh
     KEAPI_INT32 cycleCount;            // Number of charge/discharge cycles
experienced during lifetime
} BATTERY_INFO, *PBATTERY_INFO;
```

2.7.3 KEApiGetBatteryState

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetBatteryState (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 batteryNr,
      PBATTERY_STATE pBatteryState
);
```

Description:

Provides information about selected battery.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*batteryNr*	Requested battery's number. Numbers start with zero
out	*pBatteryState*	Pointer to a BATTERY_STATE structure • POWER_STATE_CHARGING, 0 • POWER_STATE_CHARGED, 1 • POWER_STATE_DISCHARGING, 2

Structure used:

```
typedef struct BatteryState
{
     KEAPI_INT32 powerState;      // Current power state - charging = 0,
charged = 1, discharging = 2
     KEAPI_INT32 fullBatteryRemainingTime;      // Remaining time in seconds
when battery is full and AC power unplugged
     KEAPI_INT32 remainingTime;   // Remaining time in seconds
     KEAPI_INT32 remainingCapacity; // Remaining capacity in mAh
     KEAPI_INT32 currentVoltage;  // Current voltage in mV
     KEAPI_INT32 rate;            // Current charging/discharging rate in mA
```

```
    KEAPI_INT32 chargeState;     // Battery charge state in percentage
} BATTERY_STATE, *PBATTERY_STATE;
```

2.8 Temperature sensors

2.8.1 KEApiGetTempSensorCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetTempSensorCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pTempSensorCount
);
```

Description:

Provides number of temperature sensors.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pTempSensorCount*	Pointer to number of installed temperature sensors

2.8.2 KEApiGetTempSensorList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetTempSensorList (
      KEAPI_UINT32 boardHandle,
      PTEMP_SENSOR pTempSensors,
      KEAPI_INT32 tempSensorCount
);
```

Description:

Provides information about temperature sensors (name and current value in millidegrees), stored in the TEMP_SENSOR structure.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*tempSensorCount*	Number of temperature sensors
out	*pTempSensors*	Pointer to the temperature sensors structure

Structure used:

```
typedef struct Temp_Sensor
{
     KEAPI_CHAR  name[MAX_STR];   // Sensor's name
     KEAPI_INT32 value;      // Value obtained from sensor in millidegrees
} TEMP_SENSOR, *PTEMP_SENSOR;
```

2.8.3 KEApiGetTempSensorValue

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetTempSensorValue (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 tempSensorNr,
      KEAPI_PINT32 pTempSensorValue
);
```

Description:

Derives information about current value of a temperature sensor with a given ID.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*tempSensorNr*	Number (index) of a temperature sensor. Numbers start with 0
out	*pTempSensorValue*	Pointer to the value of a given temperature sensor

2.8.4 KEApiGetCpuTemp

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetCpuTemp (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 cpuNr,
      KEAPI_PINT32 pCpuTemp
);
```

Description:

Provides information about actual temperature of the selected CPU.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*cpuNr*	Selected CPU number. CPU numbers start with zero
out	*pCpuTemp*	Pointer to a variable that receives selected CPU's temperature in millidegrees.

2.8.5 KEApiGetChipsetTemp

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetChipsetTemp (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pChipsetTemp
);
```

Description:

Provides information about actual chipset temperature.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pChipsetTemp*	Pointer to a variable that receives chipset temperature in millidegrees

2.8.6 KEApiGetSystemTemp

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetSystemTemp (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pSystemTemp
);
```

Description:

Provides information about actual system temperature.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of selected board
out	*pSystemTemp*	Pointer to a variable that receives system temperature in millidegrees.

2.9 Voltage sensors

2.9.1 KEApiGetVoltageSensorCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetVoltageSensorCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pVoltageSensorCount
);
```

Description:

Provides number of voltage sensors.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pVoltageSensorCount*	Pointer to the voltage sensors count

2.9.2 KEApiGetVoltageSensorList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetVoltageSensorList (
      KEAPI_UINT32 boardHandle,
      PVOLTAGE_SENSOR pVoltageSensors,
      KEAPI_INT32 voltageSensorCount
);
```

Description:

Provides information about voltage sensors in the structure **VOLTAGE_SENSOR**.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*voltageSensorCount*	Number of sensors, obtained from the function **KEApiGetVoltageSensorCount**
out	*pVoltageSensors*	Pointer to the voltage sensors structure **VOLTAGE_SENSOR**

Structure used:

```
typedef struct Voltage_Sensor
{
    KEAPI_CHAR  name[MAX_STR];      // Sensor's name
    KEAPI_INT32 value;  // Value obtained from the sensor in millivolts
} VOLTAGE_SENSOR, *PVOLTAGE_SENSOR;
```

2.9.3 KEApiGetVoltageSensorValue

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetVoltageSensorValue (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 voltageSensorNr,
      KEAPI_PINT32 pVoltageSensorValue
);
```

Description:

Provides current voltage value of a selected voltage sensor.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*voltageSensorNr*	Number (index) of a voltage. Numbers start with 0
out	*pVoltageSensorValue*	Pointer to the value of the selected sensor, in millivolts

2.10 Fan Control

2.10.1 KEApiGetFanCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetFanCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pFanCount
);
```

Description:

Provides number of fans.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pFanCount*	Pointer to number of fans

2.10.2 KEApiGetFanList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetFanList (
      KEAPI_UINT32 boardHandle,
      PFAN pFans,
      KEAPI_INT32 fanCount
);
```

Description:

Provides information about fans.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*fanCount*	Number of fans, which can be obtained from the **KEApiGetFanCount** function
out	*pFans*	Pointer to **PFAN** structure

Structure used:

```
typedef struct Fan
{
     KEAPI_CHAR   name[MAX_STR];  // Fan name
     KEAPI_UINT32 rpm;            // Revolutions per minute
     KEAPI_UINT8  pwm;            // Pulse width modulation fan control
     KEAPI_UINT8  powerMode;      // Fan speed control mode 0 = DC, 1 = PWM
     KEAPI_UINT8  method;         // Method of control - Maximal speed,
manual control or thermal cruise
} FAN, *PFAN;
```

2.10.3 KEApiGetFanSpeed

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetFanSpeed (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 fanNr,
      KEAPI_PUINT8 pwm,
      KEAPI_PINT32 rpm
);
```

Description:

Provides information about fan speed.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*fanNr*	Number (index) of a fan you are interested in. Fan numbers start with 0
out	*pwm*	Fan speed modulation pwm. Can be set to NULL if not wanted.
out	*rpm*	Returned fan speed in rpm. Can be set to NULL if not wanted.

2.10.4 KEApiGetFanMode

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetFanMode (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 fanNr,
      KEAPI_PUINT8 fanMode
);
```

Description:

Provides information about fan mode.

- Maximal speed: FAN_METHOD_MAX_SPEED, 0
- Manual control: FAN_METHOD_MANUAL, 1
- Thermal cruise: FAN_METHOD_THERMAL_CRUISE, 2.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*fanNr*	Number (index) of a fan. Fan numbers start with 0.
out	*fanMode*	Current fan mode (max speed, manual control, thermal cruise).

2.10.5 KEApiSetFanModeManual

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetFanModeManual (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 fanNr,
      KEAPI_UINT8 targetPwm
);
```

Description:

Sets the fan into manual control mode.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*fanNr*	Number (index) of a fan. Fan numbers start with 0.
in	*targetPwm*	Required fan speed modulation pwm in a range of 0 (min) - 255 (full speed)

2.10.6 KEApiSetFanModeThermalCruise

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetFanModeThermalCruise (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 fanNr,
      KEAPI_UINT32 targetTemp,
      KEAPI_UINT32 toleranceTemp
);
```

Description:

Sets the fan into thermal cruise mode.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*fanNr*	Number (index) of a fan you are interested in. Fan numbers start with 0.
in	*targetTemp*	What temperature should not be exceeded, in millidegrees Celsius
in	*toleranceTemp*	Tolerance of the target temperature. The controlled temperature should fluctuate around the target temperature within given tolerance, i.e. If the target temperature is 30 degrees Celsius, and the tolerance is set to 2 degrees, then the temperature is kept in the range of 28 - 32 degrees.

2.10.7 KEApiGetCpuFanSpeed

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetCpuFanSpeed (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pCpuFanRpm
);
```

Description:

Provides CPU fan speed (revolutions per minute).

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pCpuFanRpm*	Pointer to a variable that receives CPU fan rpm

2.10.8 KEApiSetCpuFanSpeed

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetCpuFanSpeed (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT8 cpuFanSpeedPwm
);
```

Description:

Sets speed of CPU fan using Pulse Width Modulation.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*cpuFanSpeedPwm*	Required CPU fan speed pwm 0 (min) - 255 (max. speed)

2.10.9 KEApiGetSystemFanSpeed

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetSystemFanSpeed (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pSystemFanRpm
);
```

Description:

Provides system fan speed (revolutions per minute).

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pSystemFanRpm*	Pointer to a variable that receives system fan rpm

2.10.10 KEApiSetSystemFanSpeed

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetSystemFanSpeed (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT8 systemFanSpeedPwm
);
```

Description:

Sets speed of system fan using Pulse Width Modulation.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of selected board
in	*systemFanSpeedPwm*	Required system fan speed pwm 0 (min) - 255 (max. speed)

2.11 Display

2.11.1 KEApiGetDisplayCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetDisplayCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pDisplayCount
);
```

Description:

Provides number of installed displays.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pDisplayCount*	Number of installed displays

2.11.2 KEApiGetBacklightValue

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetBacklightValue (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 displayNr,
      KEAPI_PINT32 pBacklight
);
```

Description:

Provides information about current backlight intensity of the selected display.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*displayNr*	Requested display's number. Numbers start with zero
out	*pBacklight*	Pointer to variable that receives actual brightness intensity in percentage

2.11.3 KEApiSetBacklightValue

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSetBacklightValue (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 displayNr,
      KEAPI_INT32 backlight
);
```

Description:

Sets backlight intensity of selected LVDS display.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*displayNr*	Requested display's number. Numbers start with zero
out	*backlight*	Required backlight intensity in percentage (0 - 100)

2.12 Network and PCI devices

2.12.1 KEApiGetNetworkDeviceCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetNetworkDeviceCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pNetworkDeviceCount
);
```

Description:

Provides a number of installed network devices.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pNetworkDeviceCount*	Number of installed network devices

2.12.2 KEApiGetNetworkDeviceList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetNetworkDeviceList (
      KEAPI_UINT32 boardHandle,
      PNETWORK_DEVICE pNetworkDevices,
      KEAPI_INT32 networkDeviceCount
);
```

Description:

Provides information about installed network devices.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*networkDeviceCount*	Number of network devices
out	*pNetworkDevices*	Pointer to array of NETWORK_DEVICE structures. The array must be preallocated as *NetworkDeviceCount* * sizeof(NETWORK_DEVICE), where *NetworkDeviceCount* is obtained from calling **KEApiGetNetworkDeviceCount**.

Structure used:

```
typedef struct Network_Device
{
     KEAPI_CHAR  ip[MAX_STR];     // IP address
     KEAPI_CHAR  mac[MAX_STR];    // MAC address in format XX-XX-XX-XX-XX-XX
     KEAPI_INT32 speed;           // Connection speed
     KEAPI_CHAR  deviceName[MAX_STR];     // Name of the network device
} NETWORK_DEVICE, *PNETWORK_DEVICE;
```

2.12.3 KEApiGetPciDeviceCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetPciDeviceCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pPciDeviceCount
);
```

Description:

Provides a number of installed PCI devices.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pPciDeviceCount*	Number of installed PCI devices

2.12.4 KEApiGetPciDeviceList

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetPciDeviceList (
      KEAPI_UINT32 boardHandle,
      PPCI_DEVICE pPciDevices,
      KEAPI_INT32 pciDeviceCount
);
```

Description:

Provides a list of PCI devices.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*pciDeviceCount*	Number of installed PCI devices
out	*pPciDevices*	Returned array of **PCI_DEVICE** structures

Structure used:

```
typedef struct Pci_Device
{
     KEAPI_INT32 domain;               // Domain number
     KEAPI_INT32 bus;                  // Bus number
     KEAPI_INT32 slot;                 // Slot number
     KEAPI_INT32 funct;                // Function number
     KEAPI_INT32 deviceId;             // Device ID
     KEAPI_INT32 vendorId;             // Vendor ID
     KEAPI_INT32 classId;              // Class ID
     KEAPI_CHAR  deviceName[MAX_STR]; // Name of the device
     KEAPI_CHAR  vendorName[MAX_STR]; // Name of the vendor
     KEAPI_CHAR  className[MAX_STR];  // Name of the class
} PCI_DEVICE, *PPCI_DEVICE;
```

2.13 Storage area

2.13.1 KEApiGetStorageCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetStorageCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PINT32 pStorageCount
);
```

Description:

Provides number of EEPROM storage areas.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pStorageCount*	Number of available storage areas

2.13.2 KEApiGetStorageSize

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetStorageSize (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 storageNr,
      KEAPI_PINT32 pStorageSize
);
```

Description:

Provides information about EEPROM storage area's size.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*storageNr*	Number of the storage area. Number of storage areas can be obtained from the **KEApiGetStorageCount** function
out	*pStorageSize*	Pointer to variable that receives size of the selected storage area.

2.13.3 KEApiStorageRead

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiStorageRead (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 storageNr,
      KEAPI_INT32 offset,
      KEAPI_PUINT8 pData,
      KEAPI_INT32 dataLength
);
```

Description:

Reads block of bytes from selected EEPROM.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*storageNr*	EEPROM storage number (starts from 0). Number of storage areas can be obtained from **KEApiGetStorageCount** function
in	*offset*	Start address offset
out	*pData*	Pointer to buffer that receives data
in	*dataLength*	Number of bytes to read

2.13.4 KEApiStorageWrite

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiStorageWrite (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 storageNr,
      KEAPI_INT32 offset,
      KEAPI_PUINT8 pData,
      KEAPI_INT32 dataLength
);
```

Description:

Writes block of bytes to selected EEPROM.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of selected board
in	*storageNr*	EEPROM storage number. Number of storage areas can be obtained from **KEApiGetStorageCount** function
in	*offset*	Start address offset
in	*pData*	Pointer to buffer that contains data to write to EEPROM
in	*dataLength*	Number of bytes to write

2.14 I2C

2.14.1 KEApiGetI2cBusCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetI2cBusCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PUCHAR pI2cBusCount
);
```

Description:

Function for getting number of active I2C buses.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
out	*pI2cBusCount*	Pointer to the variable where the I2C bus count is saved

2.14.2 KEApiI2cWrite

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiI2cWrite (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR i2cNr,
      KEAPI_UCHAR i2cAddress,
      KEAPI_UCHAR memoryAddress,
      KEAPI_PUCHAR pData,
      KEAPI_UINT32 dataLength
);
```

Description:

This function writes data to the slave I2C device connected to the I2C bus.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*i2cNr*	Number of I2C Buses. From 0 to (*I2cBusCount* - 1) returned by **KEApiGetI2cBusCount**.
in	*i2cAddress*	Address of I2C slave device
in	*memoryAddress*	Address of register/memory (EEPROM cell address for example). Depends on device.
in	*pData*	Data to write
in	*dataLength*	Length of data to write

2.14.3 KEApiI2cRead

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiI2cRead (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR i2cNr,
      KEAPI_UCHAR i2cAddress,
      KEAPI_UCHAR memoryAddress,
      KEAPI_BOOL memoryAddressUsed,
      KEAPI_PUCHAR pData,
      KEAPI_PUINT32 pDataLength
);
```

Description:

This function reads data from the slave I2C device connected to the I2C bus.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*i2cNr*	Number of I2C Buses. From 0 to (*I2cBusCount* - 1) returned by **KEApiGetI2cBusCount**.
in	*i2cAddress*	Address of I2C slave device
in	*memoryAddress*	Address of register/memory (EEPROM cell address for example). Depends on device.
in	*memoryAddressUsed*	If TRUE – function reads data from address in *MemoryAddress* parameter. Otherwise – from a current position index (depends on device).
out	*pData*	Buffer for read data

in/ out	*pDataLength*	Also an "out" parameter. When the function finishes, this parameter contains a real value of the read data length.

2.15 SMBus

2.15.1 KEApiGetSmbusCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGetSmbusCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PUCHAR pSmbusCount
);
```

Description:

Function for getting number of active SMBuses.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
out	*pSmbusCount*	Pointer to variable to save SMBus count

2.15.2 KEApiSmbusQuickCommand

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusQuickCommand (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR operation
);
```

Description:

Quick command read/write may be used to simply turn a device on/off or to enable/disable low-power standby mode etc. There is no data received. For additional information, refer to the System Management Bus(SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*operation*	What should be done • SMBUS_WRITE (0) • SMBUS_READ (1)

2.15.3 KEApiSmbusSendByte

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusSendByte (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR byte
);
```

Description:

A simple device may accept up to 256 possible encoded commands in a form of a byte. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*byte*	Command. Depends on device

2.15.4 KEApiSmbusReceiveByte

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusReceiveByte (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_PUCHAR pByte
);
```

Description:

A simple device may have information that the host needs. It can do so with Receive byte protocol. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
out	*pByte*	Device information byte. Depends on device

2.15.5 KEApiSmbusWriteByte

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusWriteByte (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR command,
      KEAPI_UCHAR byte
);
```

Description:

This function writes data of size of byte to a device. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device

in	*command*	Command code. Depends on device
in	*byte*	Data

2.15.6 KEApiSmbusReadByte

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusReadByte (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR command,
      KEAPI_PUCHAR pByte
);
```

Description:

This function reads data of size of byte from a device. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*command*	Command code. Depends on device
out	*pByte*	Pointer to the data value

2.15.7 KEApiSmbusWriteWord

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusWriteWord (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR command,
      KEAPI_UINT16 word
);
```

Description:

This function writes data of size of word to a device. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*command*	Command code. Depends on device
in	*word*	Data word to write

2.15.8 KEApiSmbusReadWord

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusReadWord (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR command,
      KEAPI_PUINT16 pWord
);
```

Description:

This function reads data of size of word from a device. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*command*	Command code. Depends on device
out	*pWord*	Pointer to the data value

2.15.9 KEApiSmbusWriteBlock

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusWriteBlock (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR command,
      KEAPI_PUCHAR pData,
      KEAPI_UCHAR dataLength
);
```

Description:

This function writes up to 32 bytes to the device. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*command*	Command code. Depends on device
in	*pData*	Pointer to a data block of size up to 32 bytes
in	*dataLength*	Length of a data block

2.15.10 KEApiSmbusReadBlock

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiSmbusReadBlock (
      KEAPI_UINT32 boardHandle,
      KEAPI_UCHAR smbusNr,
      KEAPI_UCHAR smbusAddress,
      KEAPI_UCHAR command,
```

```
      KEAPI_PUCHAR pData,
      KEAPI_PUCHAR pDataLength
);
```

Description:

This function reads up to 32 byte from the device. For additional information, refer to the System Management Bus (SMBus) Specification Version 2.0, which is available at http://smbus.org/specs/smbus20.pdf.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*smbusNr*	Number of SMBus. From 0 to (*SmbusCount* - 1) returned by **KEApiGetSmbusCount**.
in	*smbusAddress*	Address of SMBus slave device
in	*command*	Command code. Depends on device
out	*pData*	Pointer to a data block of size up to 32 bytes
in/ out	*pDataLength*	This is also the "out" parameter. It is a pointer to the length of data to read. After completing the function, this parameter contains real value of the data length.

2.16 GPIO

2.16.1 KEApiGpioGetCount

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGpioGetCount (
      KEAPI_UINT32 boardHandle,
      KEAPI_PUINT32 pGpioCount
);
```

Description:

Function for getting number of active GPIOs.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
out	*pGpioCount*	Pointer to a variable where the GPIO count is saved

2.16.2 KEApiGpioGetDirectionCaps

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGpioGetDirectionCaps (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 gpioNr,
      KEAPI_PBOOL pIsInput,
      KEAPI_PBOOL pIsOutput
);
```

Description:

Function to get possible directions of a selected GPIO.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*gpioNr*	Number of a GPIO port. From 0 to (*GpioCount* - 1) returned

		by **KEApiGpioGetCount**.
out	*pIsInput*	If TRUE then pin can be input
out	*pIsOutput*	If TRUE then pin can be output

2.16.3 KEApiGpioGetDirection

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGpioGetDirection (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 gpioNr,
      KEAPI_PUCHAR pDirection
);
```

Description:

Function for getting current direction of selected GPIO pin.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*gpioNr*	Number of an input GPIO. From 0 to (*GpioCount* - 1) returned by **KEApiGpioGetCount**.
out	*pDirection*	Pointer to a variable where the GPIO direction (0 – in, 1- out) is saved

2.16.4 KEApiGpioSetDirection

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGpioSetDirection (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 gpioNr,
      KEAPI_UCHAR  direction
);
```

Description:

Function for setting direction of selected GPIO pin.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*gpioNr*	Number of an input GPIO. From 0 to (*GpioCount* - 1) returned by **KEApiGpioGetCount**.
in	*direction*	GPIO direction to set (0 – in, 1- out)

2.16.5 KEApiGpioGetLevel

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGpioGetLevel (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 gpioNr,
      KEAPI_PUCHAR pLevel
);
```

Description:

Function for getting level of selected GPIO pin.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.

in	*gpioNr*	Number of an input GPIO. From 0 to (*GpioCount* - 1) returned by **KEApiGpioGetCount**.
out	*pLevel*	Pointer to a variable where the GPIO level (0 – low, 1- high) is saved

2.16.6 KEApiGpioSetLevel

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiGpioSetLevel (
      KEAPI_UINT32 boardHandle,
      KEAPI_UINT32 gpioNr,
      KEAPI_UCHAR level
);
```

Description:

Function for setting level of selected GPIO pin.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board.
in	*gpioNr*	Number of an output GPIO. From 0 to (*GpioCount* - 1) returned by **KEApiGpioGetCount**.
in	*level*	Level to be set (0 - low, 1 - high)

2.17 Watchdog

2.17.1 KEApiWatchdogEnable

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiWatchdogEnable (
      KEAPI_UINT32 boardHandle,
      KEAPI_INT32 timeout
);
```

Description:

Enables the watchdog.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board
in	*timeout*	Watchdog timeout interval in milliseconds

2.17.2 KEApiWatchdogTrigger

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiWatchdogTrigger (
      KEAPI_UINT32 boardHandle
);
```

Description:

Triggers the WatchDog timer.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board

2.17.3 KEApiWatchdogDisable

```
KEAPI_CALLTYPE KEAPI_UINT32 KEApiWatchdogDisable (
      KEAPI_UINT32 boardHandle
);
```

Description:

Disables the watchdog.

Parameters:

in/out	Parameter name	Description
in	*boardHandle*	Handle of the selected board

3 About Kontron

Kontron, the global leader of embedded computing technology, designs and manufactures embedded and communications standards-based, rugged COTS and custom solutions for OEMs, systems integrators, and application providers in a variety of markets. Kontron engineering and manufacturing facilities, located throughout Europe, North America, and Asia-Pacific, work together with streamlined global sales and support services to help customers reduce their time-to-market and gain a competitive advantage. Kontron's diverse product portfolio includes: boards & mezzanines, Computer-on-Modules, HMIs & displays, systems & platforms, and rugged & custom capabilities.

Kontron is a Premier member of the Intel® Embedded and Communications Alliance and has been a VDC Platinum Vendor for Embedded Computer Boards 5 years running.

Kontron is listed on the German TecDAX stock exchange under the symbol "KBC". For more information, please visit: www.kontron.com

CORPORATE OFFICES

Europe, Middle East & Africa
Oskar-von-Miller-Str. 1
85386 Eching/Munich
Germany
Tel.: +49 (0)8165/ 77 777
Fax: +49 (0)8165/ 77 219
info@kontron.com

North America
14118 Stowe Drive
Poway, CA 92064-7147
USA
Tel.: +1 888 294 4558
Fax: +1 858 677 0898
sales@us.kontron.com

Asia Pacific
17 Building,Block #1,ABP.
188 Southern West 4th Ring Road
Beijing 100070, P.R.China
Tel.: + 86 10 63751188
Fax: + 86 10 83682438
kcn@kontron.cn